# Projektowanie obiektowe

Design patterns part II

June 2020

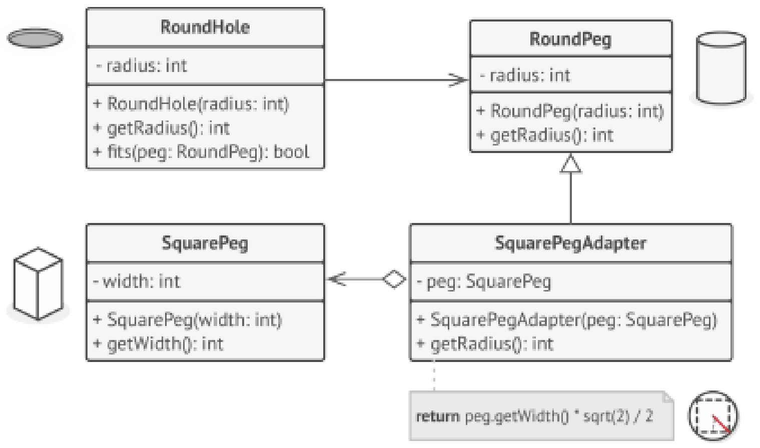
## 1 Description

The laboratory's purpose is to become familiar with structural and behavioral design patterns.

## 2 Tasks

In each task an UML scheme is presented. Use it to implement the application, which use specified design pattern. The key components (e.g. Editor in Command pattern) should be mocked. It is not advised to create an actual UI (in adapter and decorator pattern there are rather simple components presented). The most important thing is to implement the pattern mechanism.

### 2.1 Adapter



Rysunek 1: Adapting square pegs to round holes.

Tips:

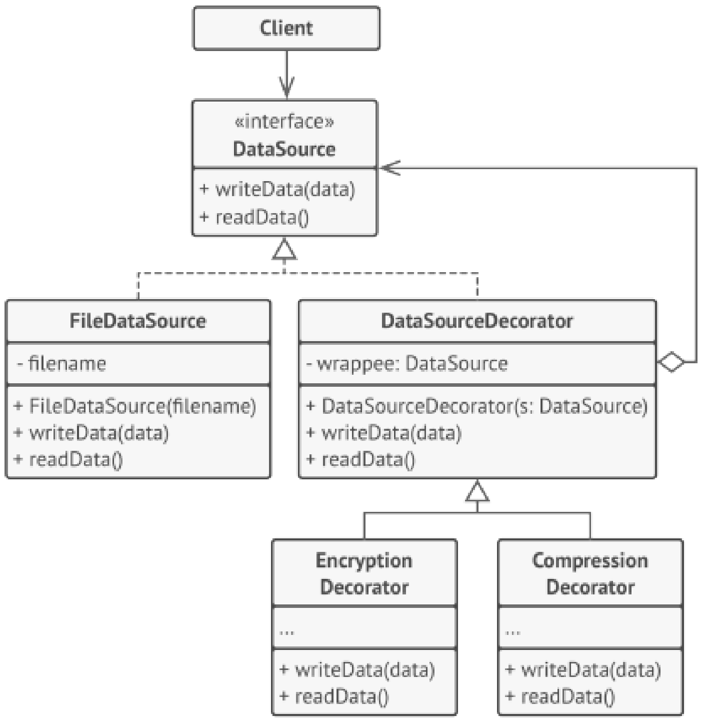
1. Make sure that you have at least two classes with incompatible interfaces (a useful service class, which you can't change, one or several client classes that would benefit from using the service class),

2. Declare the client interface and describe how clients communicate with the service.
3. Clients should use the adapter via the client interface. This will let you change or extend the adapters without affecting the client code.
4. Client interface should be as easy as possible in this case (task).

In Client code following lines should be invoked (similar to it):

```
    hole = new RoundHole(5)
    rpeg = new RoundPeg(5)

    hole.fits(rpeg) // true

    small_sqpeg = new SquarePeg(5)
    large_sqpeg = new SquarePeg(10)

    hole.fits(small_sqpeg) // this won't compile (incompatible types)

    small_sqpeg_adapter = new SquarePegAdapter(small_sqpeg)
    large_sqpeg_adapter = new SquarePegAdapter(large_sqpeg)

    hole.fits(small_sqpeg_adapter) // true
    hole.fits(large_sqpeg_adapter) // false
```

## 2.2 Decorator

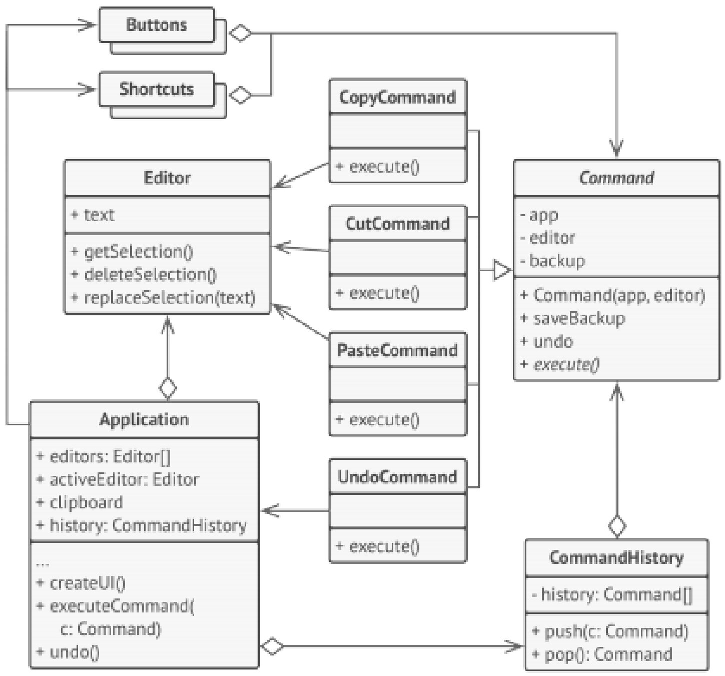


Rysunek 2: The encryption and compression decorators example.

Tips:

1. Just before the data is written to disk, the decorators encrypt and compress it. The original class writes the encrypted and protected data to the file without knowing about the change.
2. Right after the data is read from disk, it goes through the same decorators, which decompress and decode it
3. The decorators and the data source class implement the same interface, which makes them all interchangeable in the client code.
4. Use whatever encryption algorithm you want. It can be implemented from scratch (but you can use shipped implementations).

## 2.3 Command



Rysunek 3: Undoable operations in a text editor.

Tips:

1. In above example, the Command pattern helps to track the history of executed operations and makes it possible to revert an operation if needed.
2. The client code (GUI elements, command history, etc.) isn't coupled to concrete command classes because it works with commands via the command interface.

Simple example of how command pattern should be executed on the client side (not connected to task - just an example).

```
    TextFileOperationExecutor textFileOperationExecutor = new TextFileOperationExecutor();
    TextFile textFile = new TextFile("file1.txt");
    textFileOperationExecutor.executeOperation(textFile::open);
    textFileOperationExecutor.executeOperation(textFile::save);
```